Test Strategy & Planning:

1. Testing Scope and Objective:

- To ensure that X.509 certificates are correctly generated, validated, and used for secure communication, preventing unauthorized access and ensuring data integrity.
- The integrity of an X.509 certificate ensures that it is authentic, unaltered, and secure for establishing trusted communication. If integrity checks fail, it can lead to security breaches, data theft, and unauthorized access.

2. Manual vs. Automated Testing:

The list of cases which can tested manually:

1. **Misconfigurations check (e.g., Wildcard Certificates, Key reuse):**
    - Some misconfigurations may be environment-specific, making automation unreliable.
        i. Example: Checking if wildcard certificates (*.example.com) are used in environments where they are not allowed.

2. **Checking Certificate Fields for Business-Specific Requirements:**
    - Business-specific fields (e.g., Organization Name, Common Name) may require human verification.
    - Some attributes (like OU=QA) may be dynamically assigned based on context.

3. **Revoked Certificate Handling:**
    - By revoking a certificate and observing behavior in browsers or APIs.
        i. Example: Accessing a web service with a revoked certificate and verifying browser behavior.

4. **Certificate Expiry Handling (User Experience Testing)**
    - Testing how an application reacts when an expired certificate is encountered (e.g., does it display a warning or prevent access?)

The list of cases which can be automated:

1. **Format Validation (PEM, DER, PKCS#12)**
    - Checking file formats is structured and repetitive, making it ideal for automation.

2. **Certificate Field Validation (Issuer, Subject, Validity, Extensions)**
    - Automating field extraction using automation saves time and ensures consistent validation.

3. **Cryptographic Algorithm Validation (RSA, ECDSA, Key Length)**
    - Checking which type of strong cryptographic algorithms (RSA 2048+, ECDSA) are used can be checked programmatically instead of checking it manually.
    - Detecting weak algorithms is rule-based and repetitive, making it perfect for automation. Can integrate with CI/CD pipelines to fail builds if weak algorithms are found.

4. **Expired Certificate Detection**
    - Certificates should not expire unnoticed, so periodic checks should be automated, to handle this we can automate alerts to notify teams before expiration.

3. Test Scenarios:

Positive Cases:

1. Verify that **cert.pem** and **key.pem** are generated successfully

® cert.pem and key.pem should be created without errors.

2. Validate the certificate format (PEM).

® The certificate should be in PEM format and readable via openssl x509 -in cert.pem -text -noout.

3. Verify that the private key is not encrypted due to -nodes option.

® key.pem should not require a password.

4. Check that the certificate contains expected fields (Issuer, Subject, private Key, Validity, Extensions).

5. Verify that the certificate contains the correct Subject information (CN, O, OU, ST, C).

6. Validate the cryptographic algorithm (e.g., RSA 2048/4096).

7. Verify that the private key is securely stored and does not get exposed.

Negative Cases:

8. Test with missing -nodes option to ensure the private key requires a password.

9. Check for duplicate certificate generation if the same command is executed twice.

® The new certificate should overwrite existing files or prompt for confirmation.

10. Test invalid -subj values (e.g., missing CN or wrong format) It should throw an error for incorrect subject values.

11. Verify the behavior when using an invalid key size (e.g., rsa:1024)

® It should throw an error due to weak key size.

12. Test generating a certificate without -keyout parameter.

® It should fail due to the missing private key output.

13. Verify the behavior when a certificate is expired.

14. Verify that the user should be able to access the website at the last minute of certificate expiry.

15. Verify a revoked certificate is not accepted.

16. Verify that the weak cryptographic algorithms (e.g., SHA1, MD5) are not used.

17. Validate certificates for misconfigurations (e.g., Key reuse).

18. Verify that the self-signed certificate should not be generated

4. Automation Strategy:

- To automate X.509 certificate validation, I will use TestNG for structured testing,
- Java KeyStore API for parsing certificates, and OpenSSL for generating test certificates.
- Bouncy Castle Library – Provides additional cryptographic operations for certificate handling.
- Maven/Gradle – Manages dependencies and builds automation tests efficiently.
- I will implement data-driven testing using TestNG's @DataProvider, allowing dynamic validation of multiple certificates (valid, expired, revoked, malformed).

5. Security & Performance Considerations:

a. To ensure strong cryptographic standards, I will enforce a minimum RSA key size of 2048 bits, SHA-256+ algorithms, and TLS 1.2/1.3 compliance. Weak algorithms like SHA-1, MD5, and RSA-1024 will be rejected.
b. For performance, I will conduct load testing with JMeter to simulate large-scale validations. Optimizations like caching and asynchronous validation will be applied to improve efficiency."